Table of Contents

Problem Statement

Objective of the use case and description

Data Cleaning

Removing missing values and cleaning data

Exploratory Data Analysis

Analyzing data and performing Univariate Analysis, Multivariate Analysis.

Prediction of income with below models.

Training model and testing there performance

Handling Data Imbalance

Ensure dataset balance and detect outliers. Then, implement feature engineering before training the model. Apply cross-validation and hyperparameter tuning to select the optimal model with the best hyperparameter values.

Problem Statement:

For this dataset, we will have to predict the income of an employee based on their different attributes. The prediction task is to determine whether a person makes over $50K a year or not

i.e., there are only 2 categories ($50K, <=$50K per year) on which each employee will be classified

> Mounting Google Drive for loading dataset

```
from google.colab import drive
# Mount Google Drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
csv_file_path  =  "/content/drive/MyDrive/income_evaluation.csv"
```

```
# import required libraries

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import os


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from collections import Counter

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, VotingCla
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve, train_test_split, KFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


from sklearn.svm import SVC, LinearSVC

sns.set(style='white', context='notebook', palette='deep')
```

```
df = pd.read_csv(csv_file_path, encoding='utf-8')
df
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relations
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-fa
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husb
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-fa
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husb
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	W
...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	W
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husb

Next steps: Generate code with df | View recommended plots

```
for i in df.columns:
  print(i)

print(f"we have total columns of {len(df.columns)}")
```

```
age
 workclass
 fnlwgt
 education
 education-num
 marital-status
 occupation
 relationship
 race
 sex
 capital-gain
 capital-loss
 hours-per-week
 native-country
 income
we have total columns of 15
```

Total have 14 columns of features, the last column: `income` is the classification label: `>50k, <=50k`, other features as below:

- `age`: numerical
- `workclass`: categorical
- `fnlwgt`: numerical
- `education`: categorical
- `education-num`: numerical
- `marital-status`: categorical
- `occupation`: categorical
- `relationship`: categorical
- `race`: categorical
- `sex`: categorical
- `capital-gain`: numerical
- `capital-loss`: numerical
- `hours-per-week`: numerical
- `native-country`: categorical

> 8 - Categorical features

> 6 - Numerical features

And one more things need to take note is the dataset is imbalanced, there are two classes values: `>50k` and `<=50k`.

- `>50k`: miniority class, around `25%`
- `<=50k`: majority class, around `75%`

Data Cleaning

Column names have space in the front, need to rename them.

```
df.columns= ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']
```

```
# a brief overview or summary of the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64
 11  capital_loss    32561 non-null  int64
 12  hours_per_week  32561 non-null  int64
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

From the summary above it can be seen that:

The DataFrame has 32561 rows and 15 columns There is no null value or missing information in the dataset as all columns have 32561 entries Data types include integers(int) and strings(object) The DataFrame shows there are no null values but a closer look at the loaded dataset by viewing in excel shows columns with (?) values

```
# Let's check which columns have question mark (?) as values in the DataFrame

# First convert the ? entries to nan
df.replace(r'\?', np.NaN, regex=True, inplace=True)


# Check missing values
df.isna().sum()
```

```
age                 0
workclass        1836
fnlwgt              0
education           0
education_num       0
marital_status      0
occupation       1843
relationship        0
race                0
sex                 0
capital_gain        0
capital_loss        0
hours_per_week      0
native_country    583
income              0
dtype: int64
```

The above shows that the workclass column has 1836 ? values while occupation column and native-country column both have 1843 and 583 ? values respectively.

While our DataFrame shows that we have no null values, it is worthy of note that these particular columns have unknown values represented or filled with question mark(?)

Three attributes have missing values, let's take care of them one by one.

```
df.occupation.value_counts()
```

```
occupation
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: count, dtype: int64
```

```
df.occupation.fillna(' unknown',inplace=True)
```

```
df.occupation.value_counts()
```

```
occupation
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 unknown              1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
```

```
 Armed-Forces            9
Name: count, dtype: int64
```

Replace missing values in `workclass` feature

```
1 df.workclass.value_counts()
2
```

```
workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

```
1 df.workclass.fillna(' Private',inplace=True)
2
```

```
1 df.workclass.value_counts()
2
```

```
workclass
 Private             24532
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

Replace missing values in `native_country` feature

```
1 df.native_country.value_counts()
2
```

```
native_country
 United-States                 29170
 Mexico                          643
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
 Haiti                            44
 Iran                             43
 Portugal                         37
 Nicaragua                        34
 Peru                             31
 France                           29
 Greece                           29
 Ecuador                          28
 Ireland                          24
 Hong                             20
 Cambodia                         19
 Trinadad&Tobago                  19
 Laos                             18
 Thailand                         18
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Honduras                         13
 Hungary                          13
 Scotland                         12
```

```
 Holand-Netherlands               1
Name: count, dtype: int64
```

```
df.native_country.fillna(' United-States',inplace=True)

```

```
# Check missing again values
df.isna().sum()
```

```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
income            0
dtype: int64
```

Exploratory Data Analysis

```
# Print statistical characteristics of numerical variables in the dataset

df.describe().round(2)
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
count	32561.00	32561.00	32561.00	32561.00	32561.00	32561.00
mean	38.58	189778.37	10.08	1077.65	87.30	40.44
std	13.64	105549.98	2.57	7385.29	402.96	12.35
min	17.00	12285.00	1.00	0.00	0.00	1.00
25%	28.00	117827.00	9.00	0.00	0.00	40.00
50%	37.00	178356.00	10.00	0.00	0.00	40.00
75%	48.00	237051.00	12.00	0.00	0.00	45.00
max	90.00	1484705.00	16.00	99999.00	4356.00	99.00

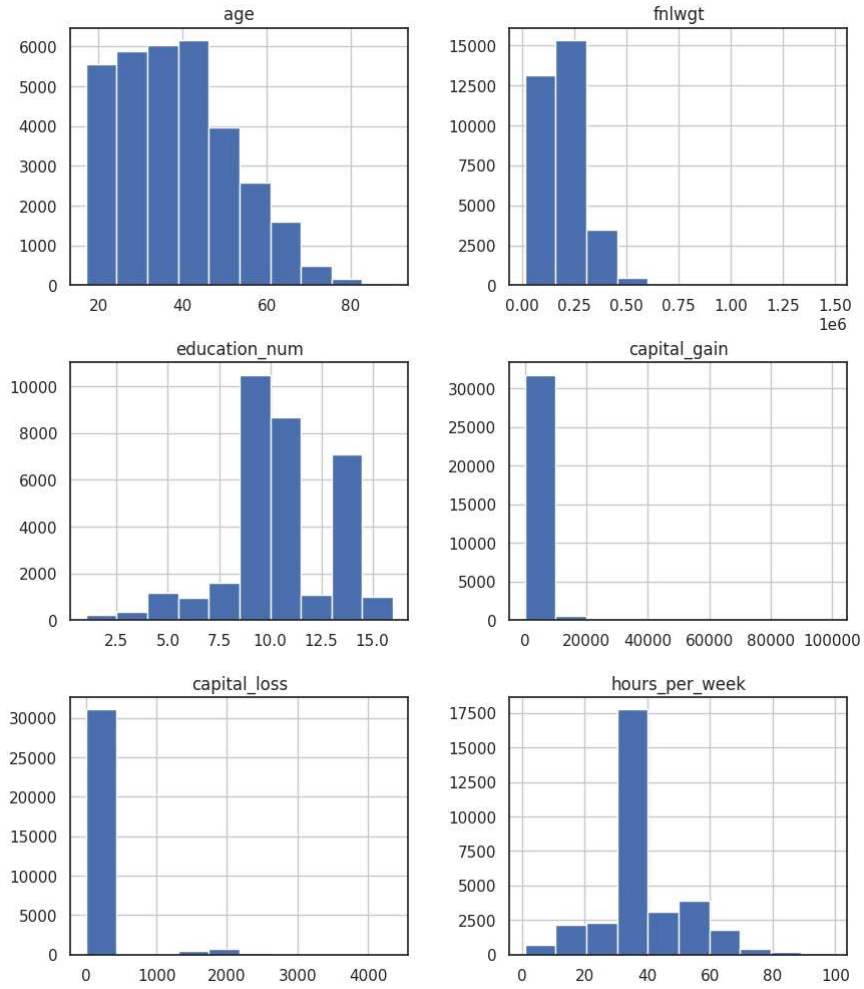
```
# Check number of unique values or categories in each column
df.nunique()
```

```
age                 73
workclass            8
fnlwgt           21648
education           16
education_num       16
marital_status       7
occupation          15
relationship         6
race                 5
sex                  2
capital_gain       119
capital_loss        92
hours_per_week      94
native_country      41
income               2
dtype: int64
```

Above are the number of distinct entries in each column. There are 73 unique values for age, 9 for workclass and so on.

Let's plot some graphs to visualize the dataset and relationship between columns

```
# plot histogram of numerical columns using pandas
df.hist(bins=10, figsize=(10,12));

```



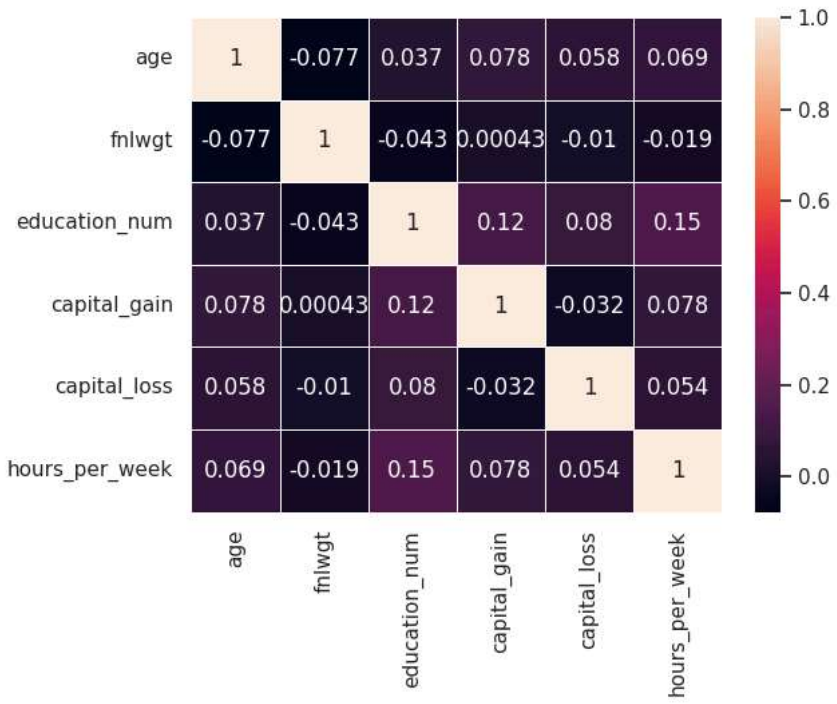
Above are Histogram plots of numerical columns in the dataset showing the distributions. we can infer from these plots that:

- Most of the people in the dataset are within the age range of `17 & 45` years of age.
- The final weight (fnlwgt) assigned to each record is majorly between 1,000,000 and 3,000,000.
- Most individuals completed between 8 to 12 years of education (educational_num).
- The profit earned from selling an asset at a price higher than its purchase cost (capital-gain) is mostly between 0–10000 USD.
- The losses from selling an asset for a price lower than the original purchase price (capital-loss) is mostly between 0–450 USD.
- Most people worked between 30–40 hours-per-week

```
1 # Correlation between numerical columns
2 num_columns = df[["age","fnlwgt","education_num","capital_gain","capital_loss","hours_per_week"]]
3 num_columns.corr()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_
age	1.000000	-0.076646	0.036527	0.077674	0.057775	
fnlwgt	-0.076646	1.000000	-0.043195	0.000432	-0.010252	-
education_num	0.036527	-0.043195	1.000000	0.122630	0.079923	
capital_gain	0.077674	0.000432	0.122630	1.000000	-0.031615	
capital_loss	0.057775	-0.010252	0.079923	-0.031615	1.000000	
hours_per_week	0.068756	-0.018768	0.148123	0.078409	0.054256	

```
# Plot correlation matrix heatmap using Seaborn
sns.heatmap(num_columns.corr(), annot=True, linewidth=0.5);
```



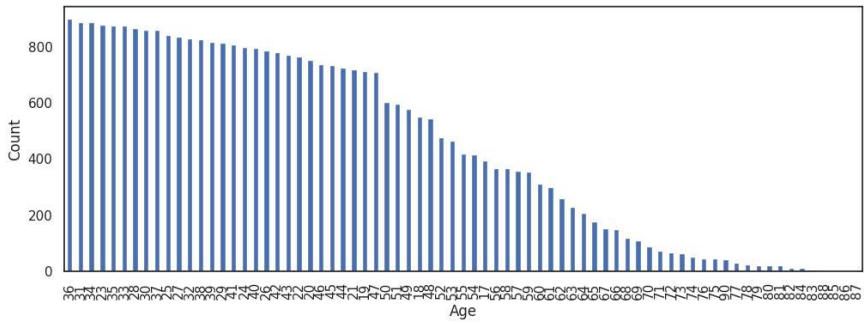
The correlation between the numerical columns is low and as such we do not have high collinearity between numerical columns

Univariate analysis

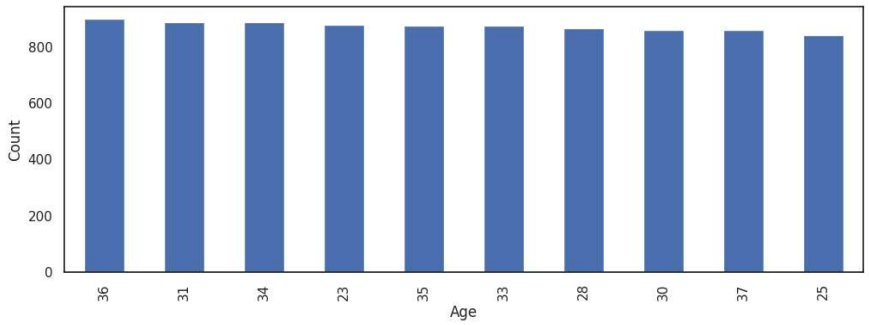
Lets visualize the count plot of unique values in each categorical column

Age

```
df["age"].value_counts().plot.bar(figsize = (12, 4));
plt.xlabel("Age")
plt.ylabel("Count")
plt.show();
```

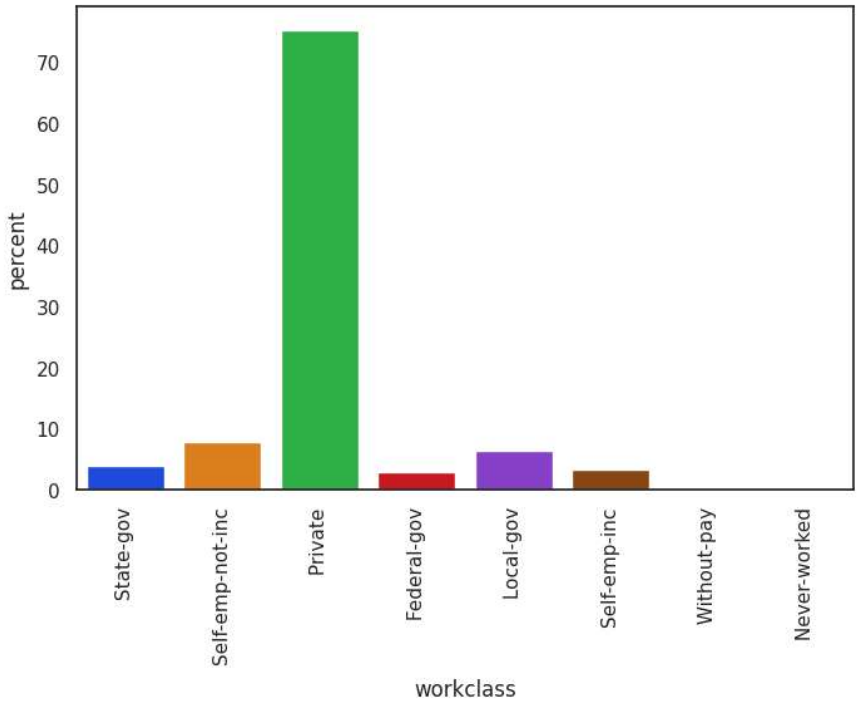



```
# Top 10 most frequent age groups in the dataset
df["age"].value_counts().head(10).plot.bar(figsize = (12, 4));
plt.xlabel("Age")
plt.ylabel("Count")
plt.show();
```



Workclass

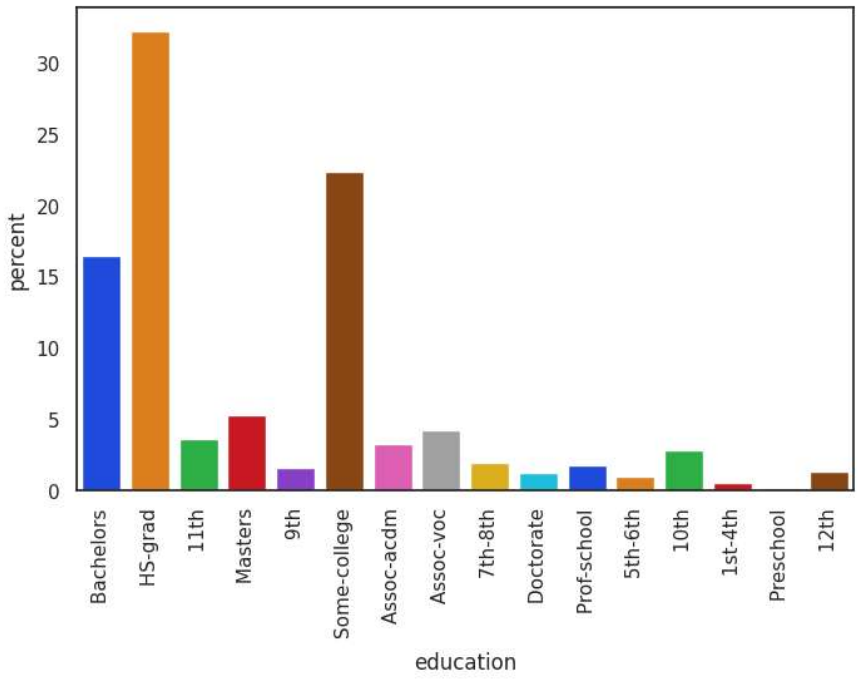
```
# Seaborn has six variations of matplotlib's palette, called deep, muted, pastel, bright, dark, and colorblind
fig, ax = plt.subplots(figsize = (8, 5))
sns.countplot(data=df, x="workclass", stat="percent", hue="workclass", palette="bright", ax=ax)
ax.tick_params(axis='x', labelrotation=90)
```



From above we can see that 70% of workclass fall under the private

Education

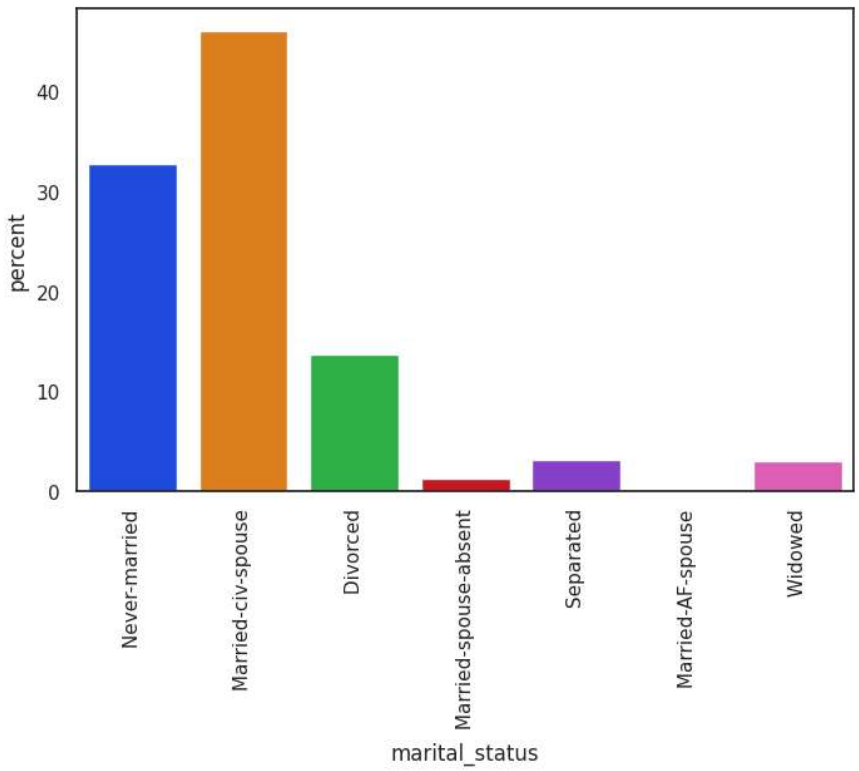
```
fig, ax = plt.subplots(figsize = (8, 5))
sns.countplot(data=df, x="education", stat="percent", hue="education", palette="bright", ax=ax)
ax.tick_params(axis='x', labelrotation=90)
```



Majority of the workclass are High school grads, bachelors degree holders and college grads

Marital-status

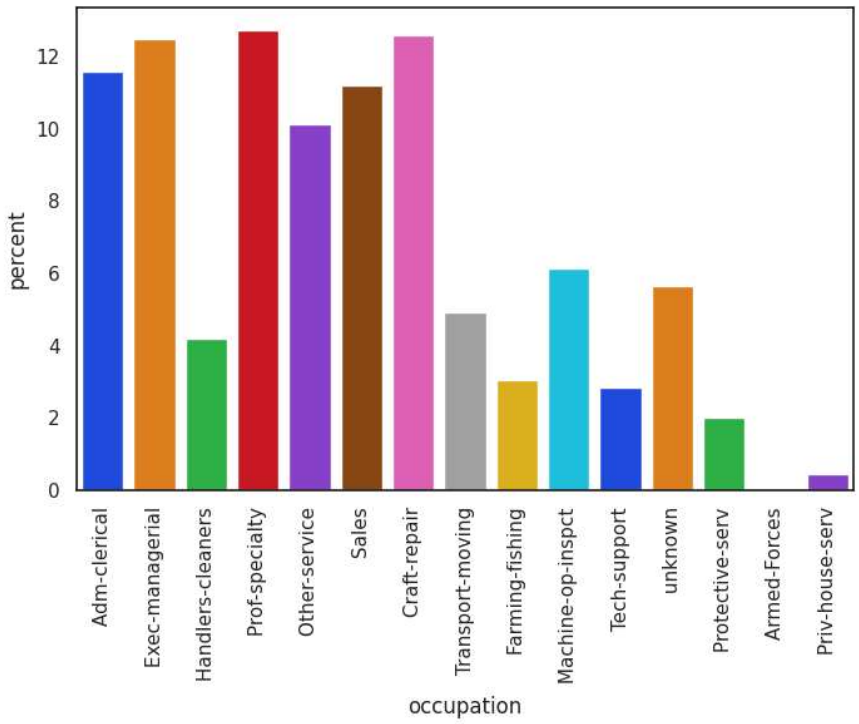
```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="marital_status", stat="percent", hue="marital_status", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



from above plot we can see `married-civ-spouse` and `never-married` having highest count comparing to rest

Occupation

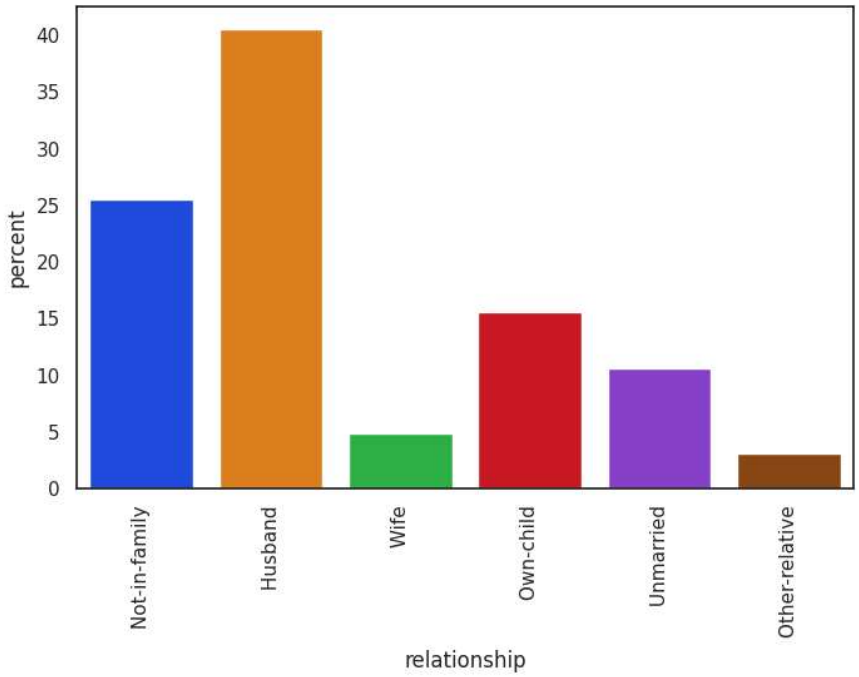
```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="occupation", stat="percent", hue="occupation", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



Majority of the adults work in the prof-specialty, craft-repair, adm-clerical, exec-managerial, sales and other-services occupation

relationship

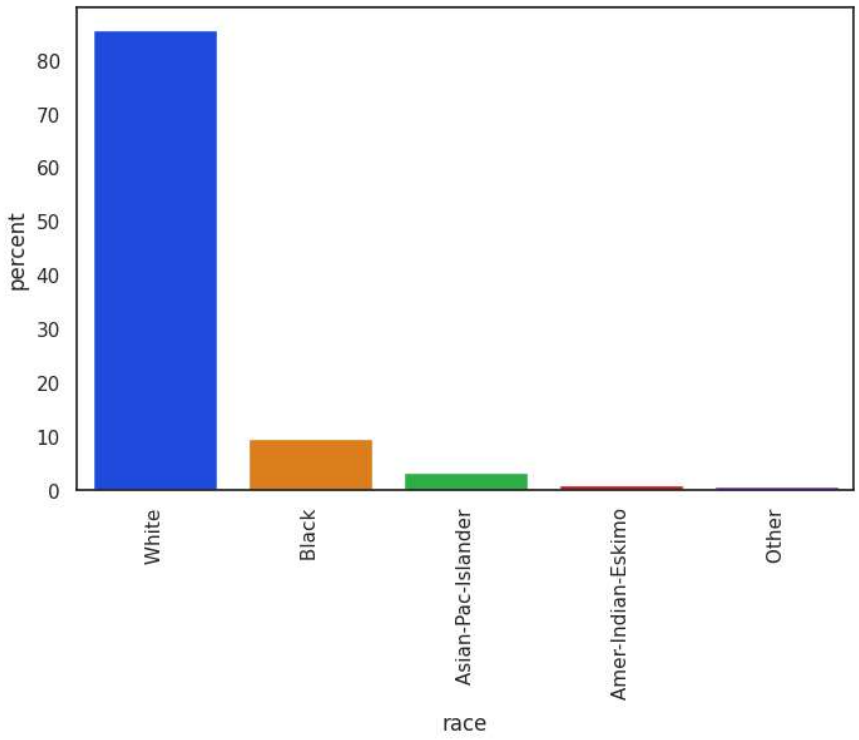
```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="relationship", stat="percent", hue="relationship", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



Majority are `husband` and `not in family` group from the above relationship

Race

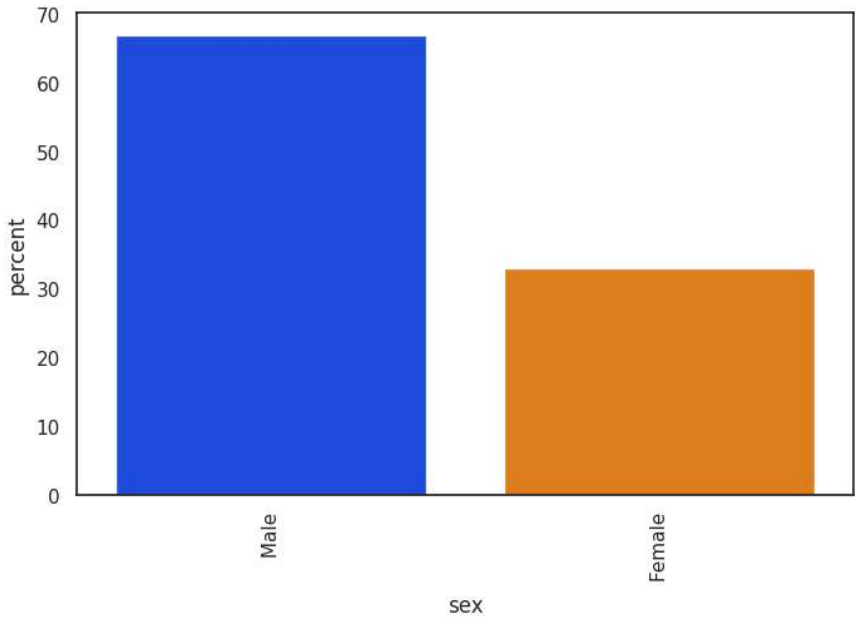
```
fig, ax = plt.subplots(figsize = (8, 5))
sns.countplot(data=df, x="race", stat="percent", hue="race", palette="bright", ax=ax)
ax.tick_params(axis='x', labelrotation=90)
```



We can see that `white` are in majority compare to any other race

Sex

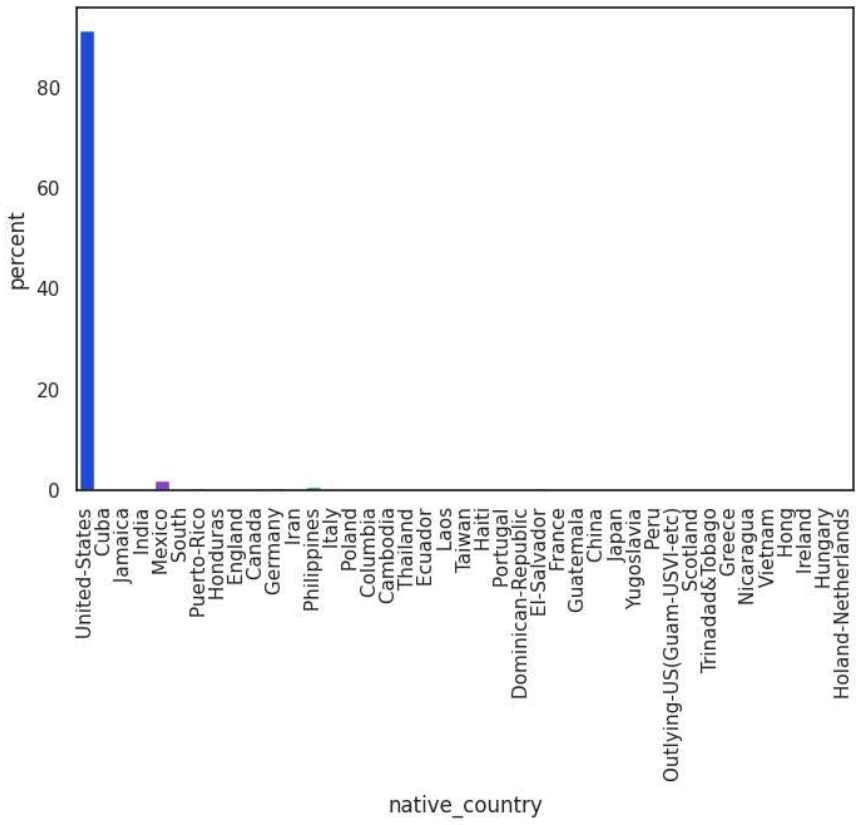
```
fig, ax = plt.subplots(figsize = (8, 5))
sns.countplot(data=df, x="sex", stat="percent", hue="sex", palette="bright", ax=ax)
ax.tick_params(axis='x', labelrotation=90)
```



Percentage of `male` are more comparing to percentage of `female`

native-country

```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="native_country", stat="percent", hue="native_country", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```

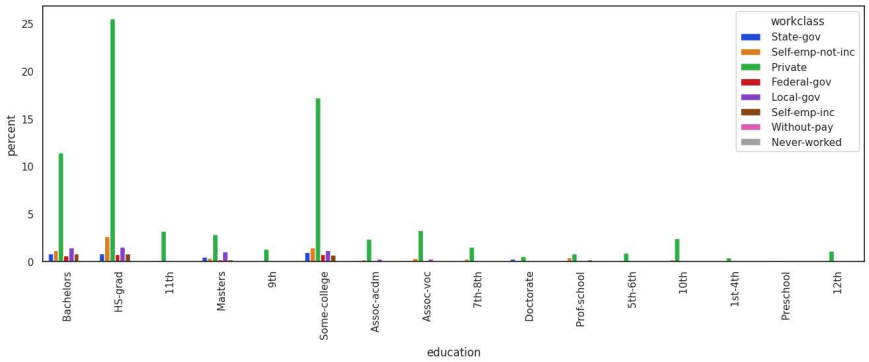


we can see high skewed in `united-states` as native country are large and other native country are very less compare to US

Multivariate analysis

Education and Workclass

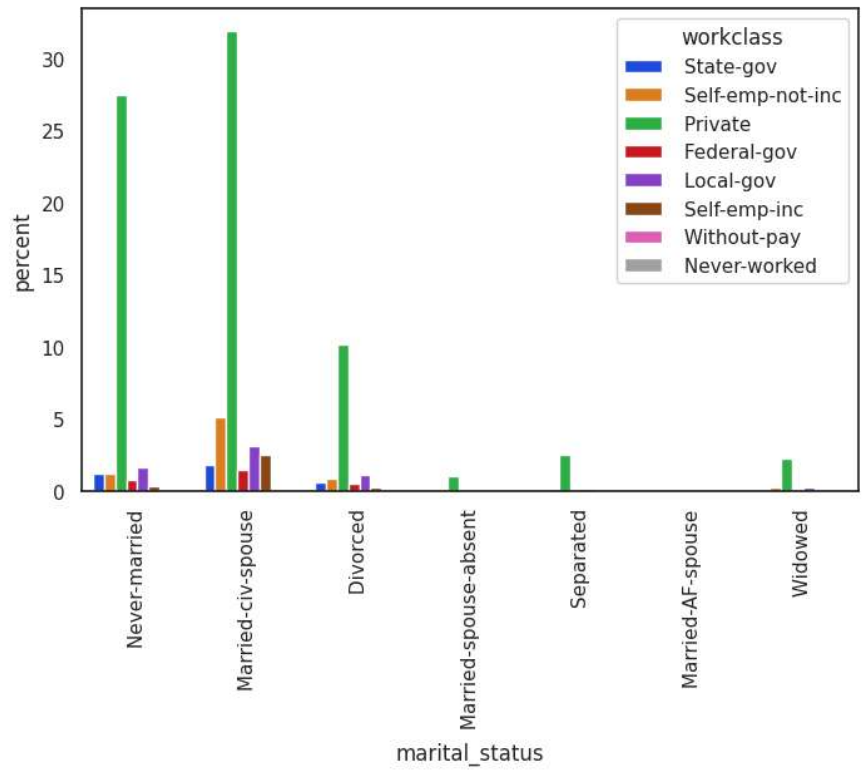
```
1 fig, ax = plt.subplots(figsize = (16, 5))
2 sns.countplot(data=df, x="education", stat="percent", hue="workclass", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



It is evident from above that majority of the workclass from different education groups work in the private sector.

Marital-status and Workclass

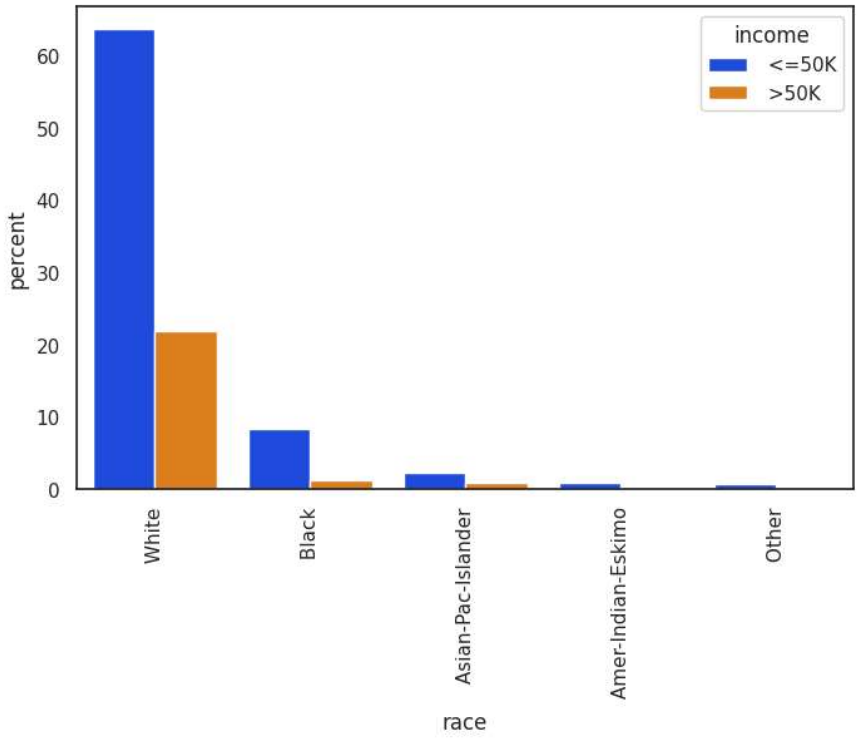
```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="marital_status", stat="percent", hue="workclass", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



Above is the distribution of workclass by marital-status. Majority of singles and married-civ-spouse work in the private sector

Race and Income

```
1 fig, ax = plt.subplots(figsize = (8, 5))
2 sns.countplot(data=df, x="race", stat="percent", hue="income", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```

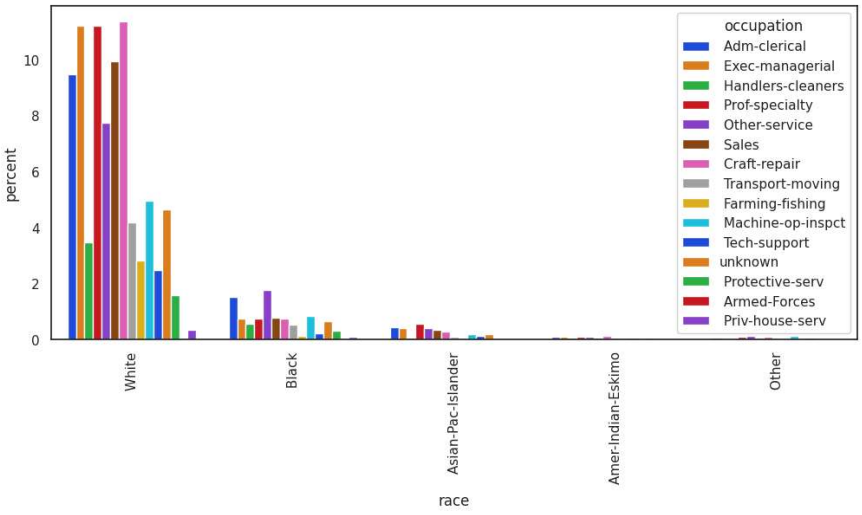


The above shows that the whites earn significantly more than other races in the dataset.

Let's explore the relationship between 'race' and 'education' and also 'occupation' to find out why.

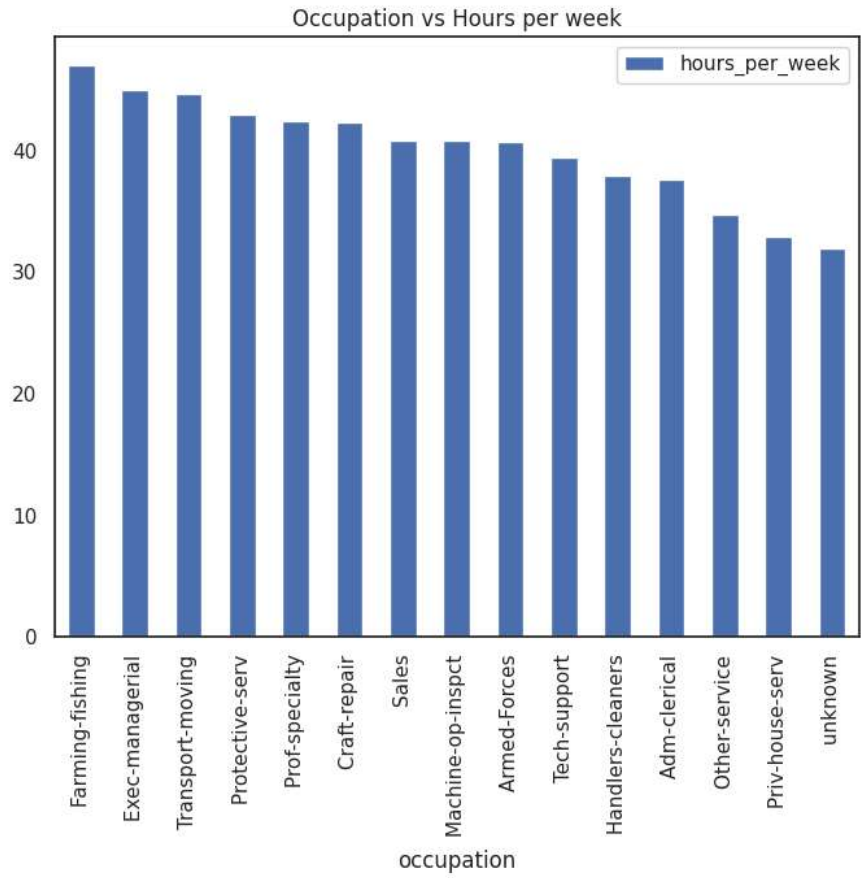
Race and Occupation

```
1 fig, ax = plt.subplots(figsize = (12, 5))
2 sns.countplot(data=df, x="race", stat="percent", hue="occupation", palette="bright", ax=ax)
3 ax.tick_params(axis='x', labelrotation=90)
```



the bar for White workers shows that the largest percentage are employed in Administrative/Clerical occupations, followed by Professional/Specialty occupations and Sales occupations comparing to other race

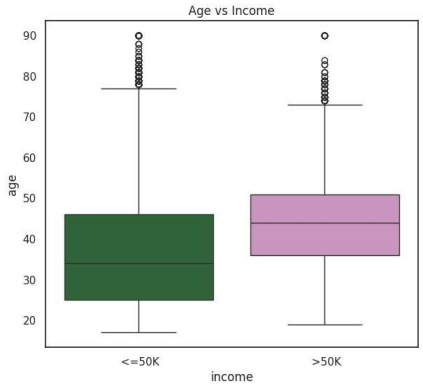
```
df1 = pd.DataFrame(df.groupby(['occupation'])['hours_per_week'].mean().sort_values(ascending = False))
df1.plot.bar(figsize=(8,6))
plt.title('Occupation vs Hours per week')
plt.show()
```

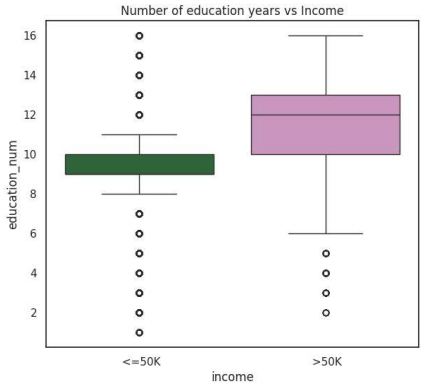


- Occupations such as farming-fishing, exec-managerial, transport-moving, professional specialty occupations, craft repairers, and machine operators, tend to have longer workweeks.
- while others, such as sales workers, handlers, cleaners, and helpers, tend to have shorter workweeks.

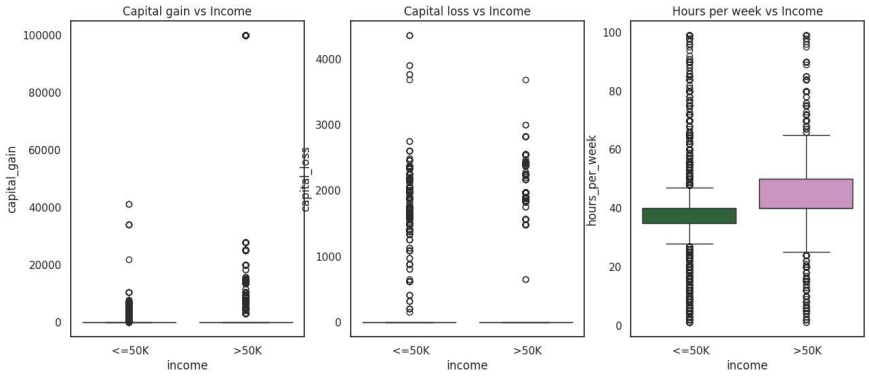
Numerical and categorical variables' relationships with target variable "income"

```
plt.figure(figsize=(15,6))

plt.subplot(1,2,1)
plt.title('Age vs Income')
sns.boxplot(x=df.income, y=df.age, palette=("cubehelix"))

plt.subplot(1,2,2)
plt.title('Number of education years vs Income')
sns.boxplot(x=df.income, y=df.education_num, palette=("cubehelix"))

plt.show()

```





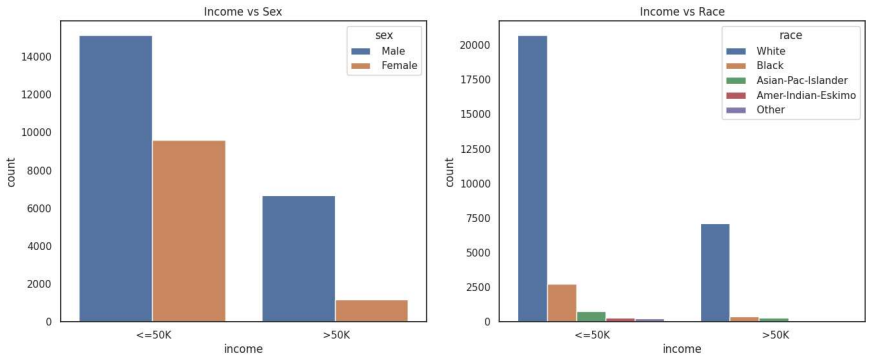

```
plt.figure(figsize=(15,6))

plt.subplot(1,3,1)
plt.title('Capital gain vs Income')
sns.boxplot(x=df.income, y=df.capital_gain, palette=("cubehelix"))

plt.subplot(1,3,2)
plt.title('Capital loss vs Income')
sns.boxplot(x=df.income, y=df.capital_loss, palette=("cubehelix"))

plt.subplot(1,3,3)
plt.title('Hours per week vs Income')
sns.boxplot(x=df.income, y=df.hours_per_week, palette=("cubehelix"))

plt.show()
```



- “Capital gain vs Income”, shows that capital gains tend to increase as income increases. There is a positive correlation between income and capital gains. This means that people with higher incomes tend to have higher capital gains.

- Capital loss vs Income" shows a weaker positive correlation between income and capital losses. Capital losses tend to be higher for those with higher incomes, however the increase is not as steep as with capital gains.
- "Hours per week vs Income" , shows a positive correlation between income and the number of hours worked per week. People who work more hours tend to earn higher incomes

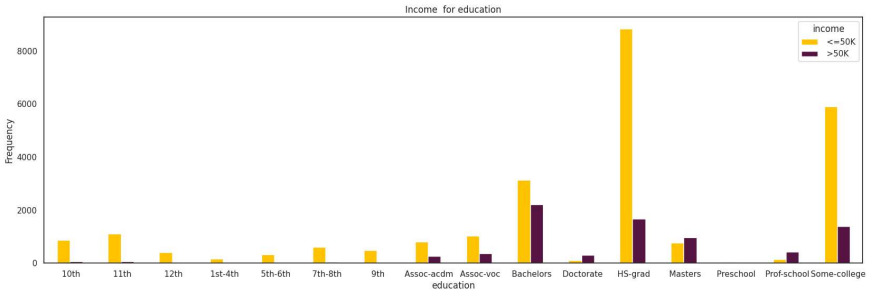
```
plt.figure(figsize=(16,6))

plt.subplot(1,2,1)
plt.title('Income vs Sex')
sns.countplot(x="income", hue="sex", data=df)

plt.subplot(1,2,2)
plt.title('Income vs Race')
sns.countplot(x="income", hue="race", data=df)

plt.show()
```



> Income Vs Sex
>
> - Men appear to earn a higher median income than women in the United States.
> - There appear to be more men than women in the higher income bracket.

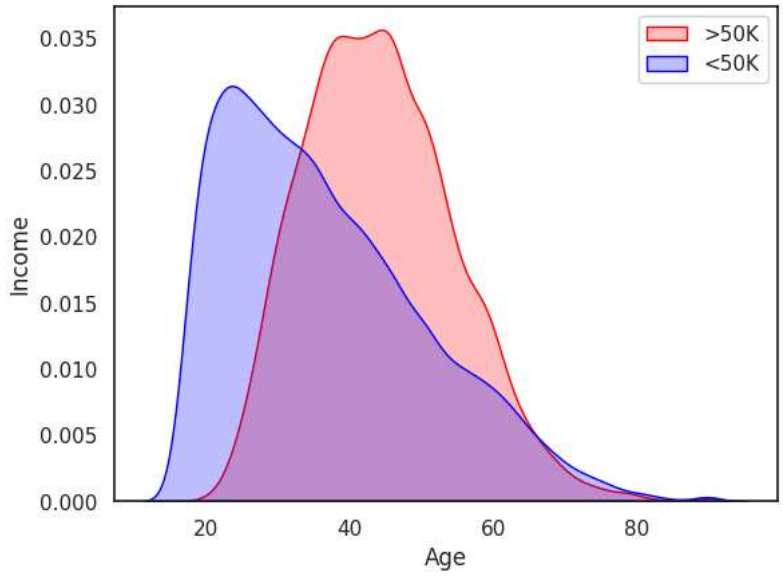
> Income Vs Race
>
> - White people appear to have the highest median income among the races depicted in the chart.
> - Black people appear to have a lower median income than White people.
> - Asian-Pacific Islanders appear to have a lower median income than White people but higher than Black people.
> - Native Americans (Amer-Indian-Eskimo) appear to have a lower median income than White people, Black people, and Asian-Pacific Islanders.
> - People identified as "Other" appear to have the lowest median income among the races depicted in the chart.

```
pd.crosstab(df['education'],df['income']).plot(kind="bar",figsize=(20,6),color=['#FFC300','#581845' ])
plt.title('Income  for education')
plt.xticks(rotation=0)
plt.ylabel('Frequency')
plt.show()
```



- People with less than a high school diploma (Preschool) appear to have the lowest frequency income.
- The frequency of income gradually increases for those with a high school diploma (HS-grad) and some college experience (Some-college).
- The income frequency appears to be the highest for those with a bachelor's degree (Bachelors).
- The income frequency shows a downward trend for those with an associate's degree (Assoc-acdm) and (Assoc-voc), a master's degree (Masters), and a professional degree (Prof-school).
- The income frequency appears to be moderate for those with a doctorate (Doctorate).

```
# Explore Age distibution
g = sns.kdeplot(df["age"][(df["income"] == ' >50K')], color="Red", shade = True)
g = sns.kdeplot(df["age"][(df["income"] == " <=50K")], ax =g, color="Blue", shade= True)
g.set_xlabel("Age")
g.set_ylabel("Income")
g = g.legend([">50K","<50K"])

```



We can see that younger people are more likely to earn less than 50K per year. Which is obvious since those people do not have any work experience. The rates of people earning more than 50K start to increase as people age into their 30s.

This is also seen in the purple line(>50K). The line starts to increase from 20 to 45 years old and then slowly decreasing.

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Model Selection - Prediction of income with below models

- Random Forest, Support Vector Machines, Logistic Regression, KNN, Decision Tree, Naive Bayes

[ ] ↳ *23 cells hidden*

Handling Data Imbalance

If the data is imbalanced, it can cause the overfitting and bias in the odel prediction. So it is important to check and cure the data imbalance if present. We check the target variable to see if it is balanced or not.

```
new_ds = df.copy(deep=True)
```

```
new_ds['income'] = new_ds['income'].str.replace('<=50K', '0')
new_ds['income'] = new_ds['income'].str.replace('>50K', '1')
new_ds['income'] = new_ds['income'].astype(np.int64)
```

```
new_ds.income.dtypes

```

```
dtype('int64')
```

We can see that, we have encoded the values of the target variable, and converted it into int data-type. This problem is a classification problem with 'Income' as the target variable. Making a copy of the dataset to work ahead

```
ds = new_ds.copy()
print(f"Unique values in 'education': {ds.education.nunique()}\nUnique values in 'Education_num': {ds['education_num'].nunique()}")
```

```
Unique values in 'education': 16
Unique values in 'Education_num': 16
```

We see that for the feature 'Education', we already have the encoded values in feature 'Education_num'. 'Education' will be removed from the dataset.

```
ds.drop(['education'], axis = 1, inplace = True)

```

'Workclass', 'Marital_status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Native_country' are the categorical variables in the data. Proper encoding or conversion of these variables is necessary for the feature engineering. We will look at these attributes and convert them one by one.

'Workclass': Starting off with the work class, we look the number of unique values and value counts for those values

```
ds.workclass.value_counts()

```

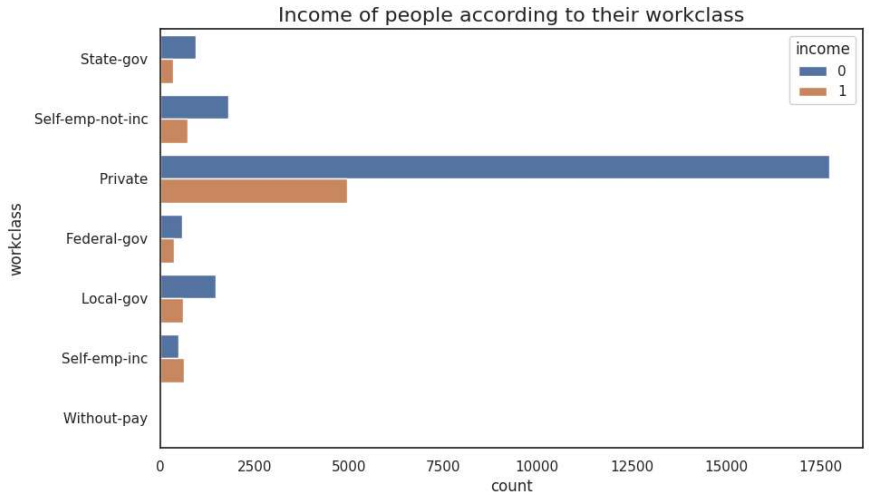
```
workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

In work class, majority of the people are private employees. The minority of people are either working without-pay or they have never-worked. We can combine the values of these two values as one.

```
ds['workclass'] = ds['workclass'].str.replace('Never-worked', 'Without-pay')

```

Now, we have 8 unique values in this feature.

```
plt.figure(figsize = (10,6))
plt.title("Income of people according to their workclass", fontsize = 16)
sns.countplot(y = ds['workclass'], hue = ds['income'])
plt.show()
```



We see that the majority of people who have income more than 50K a year are from private sector. Same goes for the people with income less than 50K. But for the Self Employed sector, the number of people whose income > 50K are more than the number of people whose income < 50K. Now, moving ahead with replacing the null values and encoding the feature. We will replace the NaN values in the 'Workclass' feature by the mode of the column, grouping it by the 'Occupation' feature. We now have 7 unique values in Workclass feature. We can encode these values using the frequency encoding technique.

```
new_ds
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty
...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial

32561 rows × 15 columns

Next steps: Generate code with new_ds | View recommended plots

```
from sklearn import preprocessing

categorical = ['workclass','education', 'marital_status', 'occupation', 'relationship','race', 'sex','native_country']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        new_ds[feature] = le.fit_transform(new_ds[feature])
```

Outliers:¶

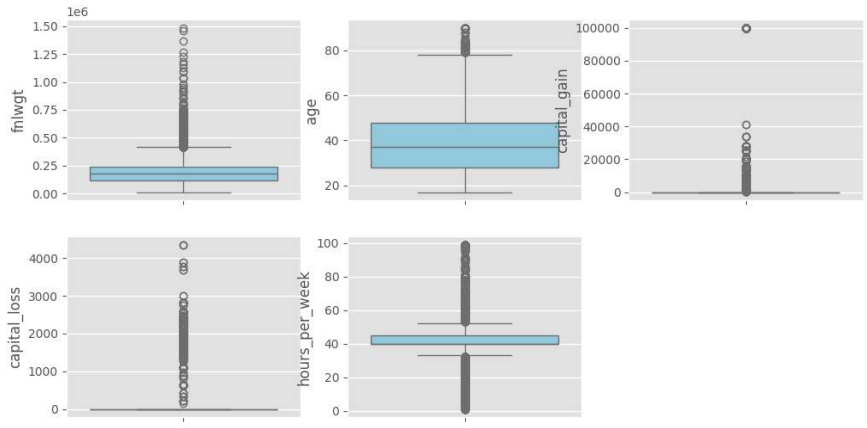
We check if any outliers are present in the continous attributes of the dataset. We check it both by visualisations and the zscore for the continous columns.

```
plt.style.use('default')

```

```
plt.style.use('ggplot')
```

```
clist = ['fnlwgt','age','capital_gain','capital_loss','hours_per_week']
plt.figure(figsize = (12,6))
for i in range(0, len(clist)):
    plt.subplot(2,3, i+1)
    sns.boxplot(ds[clist[i]], color = 'skyblue')
print("BoxPlots of the features:")
plt.show()
```

```
BoxPlots of the features:
```



Outliers are present in the continous columns of the feature. We will check the z-score of the features and and clip them from the data.

```
from scipy.stats import zscore
zabs = np.abs(zscore(new_ds.loc[:,['fnlwgt','hours_per_week']]))
print(np.shape(np.where(zabs >= 3)))
new_ds = new_ds[(zabs < 3).all(axis = 1)]
new_ds
```

(2, 787)

	age	workclass	fnlwgt	education	education_num	marital_status	occupation
0	39	6	77516	9	13	4	0
1	50	5	83311	9	13	2	3
2	38	3	215646	11	9	0	5
3	53	3	234721	1	7	2	5
4	28	3	338409	9	13	2	9
...	...	...	...	...	...	...	...
32556	27	3	257302	7	12	2	12
32557	40	3	154374	11	9	2	6
32558	58	3	151910	11	9	6	0
32559	22	3	201490	11	9	4	0
32560	52	4	287927	11	9	2	3

31781 rows × 15 columns

Next steps: Generate code with new_ds View recommended plots

WE have a total of 787 outliers in the data. After removing the outliers, we have 31781 observations left.

Scaling:¶

```
1 from sklearn.preprocessing import MinMaxScaler
2 scale = MinMaxScaler()
3 new_ds.loc[:,'age':'hours_per_week'] = scale.fit_transform(new_ds.loc[:,'age':'hours_per_week'])
4 new_ds
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	rela
	.301370	0.750	0.132035	0.600000	0.800000	0.666667	0.000000	
	.452055	0.625	0.143765	0.600000	0.800000	0.333333	0.214286	
	.287671	0.375	0.411625	0.733333	0.533333	0.000000	0.357143	
	.493151	0.375	0.450235	0.066667	0.400000	0.333333	0.357143	
	.150685	0.375	0.660111	0.600000	0.800000	0.333333	0.642857	
	...	...	...	...	...	...	...	
	.136986	0.375	0.495942	0.466667	0.733333	0.333333	0.857143	
	.315068	0.375	0.287604	0.733333	0.533333	0.333333	0.428571	
	.561644	0.375	0.282617	0.733333	0.533333	1.000000	0.000000	
	.068493	0.375	0.382972	0.733333	0.533333	0.666667	0.000000	
	.479452	0.500	0.557930	0.733333	0.533333	0.333333	0.214286	

; × 15 columns

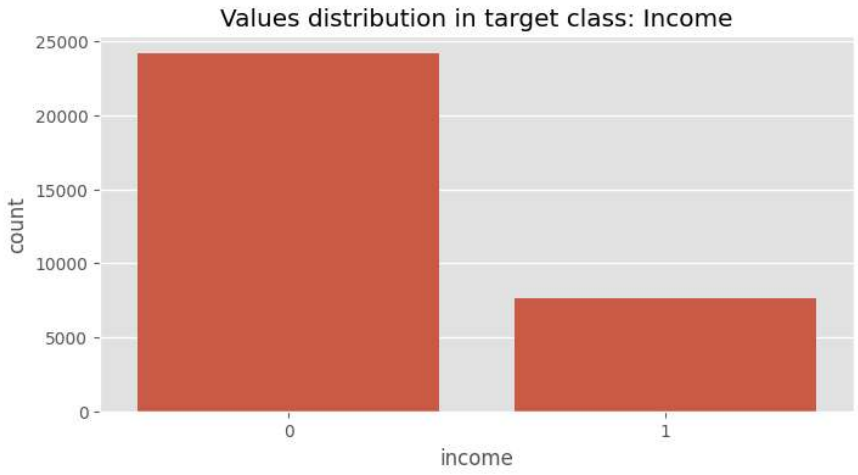
Next steps: Generate code with new_ds View recommended plots

AS we can see from the above table that the data is now more normalised and can be used by the models for learning.

Data Imbalance:

If the data is imbalanced, it can cause the overfitting and bias in the odel prediction. So it is important to check and cure the data imbalance if present. We check the target variable to see if it is balanced or not.

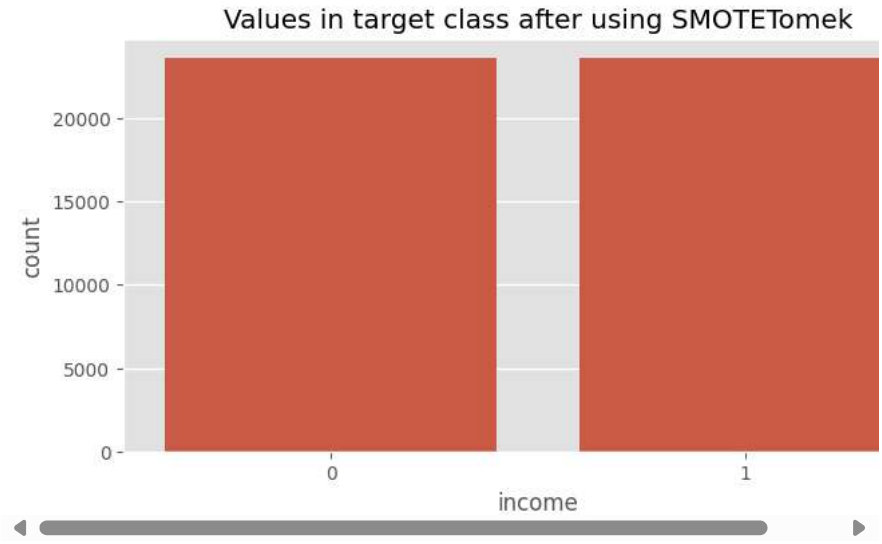
```
1 plt.figure(figsize = (8, 4))
2 plt.title("Values distribution in target class: Income")
3 sns.countplot(data = new_ds, x = 'income')
4 plt.show()
```



As we can see that data is imbalanced. In order **to remove the data imbalance, we use the SMOTETomek class to create synthetic values using KNN algorithm.**

```
from imblearn.combine import SMOTETomek
x = new_ds.loc[:,"age":"native_country"]
y = new_ds.loc[:,"income"]
smk = SMOTETomek()
x_new, y_new = smk.fit_resample(x, y)
```

```
plt.figure(figsize = (8, 4))
plt.title("Values in target class after using SMOTETomek")
sns.countplot(x = y_new)
plt.show()
```



As we can see that we now have a balanced dataset, so we can model ahead with the model building part.

Model Building:

Starting with the spliting of the training and testing data. For that, we check to see what is the best random state.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

max_accuracy = 0
best_rs = 0
for i in range(1, 150):
    x_train, x_test, y_train, y_test = train_test_split(x_new, y_new, test_size = 0.30, random_state = i)
    lg = LogisticRegression()
    lg.fit(x_train, y_train)
    pred = lg.predict(x_test)
    acc = accuracy_score(y_test, pred)
    if acc > max_accuracy: # after each iteration, acc is replace by the best possible accuracy
        max_accuracy = acc
        best_rs = i
print(f"Best Random State is {best_rs}, {max_accuracy*100}")
```

```
Best Random State is 136, 78.62849063273242
```

```
x_train, x_test, y_train, y_test = train_test_split(x_new, y_new, test_size = 0.30, random_state = 67)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Model Fitting:

Fitting 7 different models to check which model gives the best accuracy.

```
# For Logistic Regression
lg = LogisticRegression()
lg.fit(x_train, y_train)
pred_lg = lg.predict(x_test)
print("Accuracy Score of Logistic Regression model is", accuracy_score(y_test, pred_lg)*100)

# For Decision Tree Classifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
pred_dtc = dtc.predict(x_test)
print("Accuracy Score of Decision Tree Classifier model is", accuracy_score(y_test, pred_dtc)*100)

# For K-Nearest Neighbour Classifier
knc = KNeighborsClassifier(n_neighbors = 5)
knc.fit(x_train, y_train)
pred_knc = knc.predict(x_test)
print("Accuracy Score of K-Nearest Neighbour Classifier model is", accuracy_score(y_test, pred_knc)*100)

# For Support Vector Classifier
svc = SVC(kernel = 'rbf')
svc.fit(x_train, y_train)
pred_svc = svc.predict(x_test)
print("Accuracy Score of Support Vector Classifier model is", accuracy_score(y_test, pred_svc)*100)

# For Random Forest Classifier
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
pred_rfc = rfc.predict(x_test)
print("Accuracy Score of Random Forest model is", accuracy_score(y_test, pred_rfc)*100)

# For MultinomialNB
nb = MultinomialNB() # making the Multinomial Naive Bayes class
nb.fit(x_train, y_train) # fitting the model
pred_nb = nb.predict(x_test) # predicting the values
print("Accuracy Score of MultinomialNB model is", accuracy_score(y_test, pred_nb)*100)

# For ADA Boost Classifier
ada= AdaBoostClassifier()
ada.fit(x_train, y_train) # fitting the model
pred_ada = ada.predict(x_test) # predicting the values
print("Accuracy Score of ADA Boost model is", accuracy_score(y_test, pred_ada)*100)
```

```
Accuracy Score of Logistic Regression model is 77.13679745493107
Accuracy Score of Decision Tree Classifier model is 85.89607635206788
Accuracy Score of K-Nearest Neighbour Classifier model is 85.43655001767408
Accuracy Score of Support Vector Classifier model is 72.1951219512195
```

```
Accuracy Score of Random Forest model is 90.50547896783316
Accuracy Score of MultinomialNB model is 73.11417462000706
Accuracy Score of ADA Boost model is 86.07988688582539
```

Best accuracy score is given by Random Forest Classifier model. In order to avoid the bias and overfitting or underfitting, we cross validate the models and check the mean accuracy score of them.

Cross Validation:

Cross validating the m,odels to see if they are underfitting or overfitting and to prevent bias. We will compare the mean accuracy scores of the model.

```
from sklearn.model_selection import cross_val_score

lg_scores = cross_val_score(lg, x_new, y_new, cv = 10) # cross validating the model
print(lg_scores) # accuracy scores of each cross validation cycle
print(f"Mean of accuracy scores is for Logistic Regression is {lg_scores.mean()*100}\n")

dtc_scores = cross_val_score(dtc, x_new, y_new, cv = 10)
print(dtc_scores)
print(f"Mean of accuracy scores is for Decision Tree Classifier is {dtc_scores.mean()*100}\n")

knc_scores = cross_val_score(knc, x_new, y_new, cv = 10)
print(knc_scores)
print(f"Mean of accuracy scores is for KNN Classifier is {knc_scores.mean()*100}\n")

svc_scores = cross_val_score(svc, x_new, y_new, cv = 10)
print(svc_scores)
print(f"Mean of accuracy scores is for SVC Classifier is {svc_scores.mean()*100}\n")

rfc_scores = cross_val_score(rfc, x_new, y_new, cv = 10)
print(rfc_scores)
print(f"Mean of accuracy scores is for Random Forest Classifier is {rfc_scores.mean()*100}\n")

nb_scores = cross_val_score(nb, x_new, y_new, cv = 10)
print(nb_scores)
print(f"Mean of accuracy scores is for MultinomialNB is {nb_scores.mean()*100}\n")

ada_scores = cross_val_score(ada, x_new, y_new, cv = 10)
print(ada_scores)
print(f"Mean of accuracy scores is for ADA Boost Classifier is {ada_scores.mean()*100}\n")
```

```
[0.76861082 0.77518558 0.78621421 0.76882291 0.76564157 0.76861082
 0.78812301 0.7804878  0.77900318 0.77179215]
Mean of accuracy scores is for Logistic Regression is 77.52492046659599

[0.79893955 0.80742312 0.80169671 0.88441145 0.89798515 0.8911983
 0.9028632  0.90943796 0.89946978 0.89692471]
Mean of accuracy scores is for Decision Tree Classifier is 86.90349946977729

[0.85577943 0.84772004 0.85068929 0.85874867 0.86702015 0.87423118
 0.87741251 0.88313892 0.8776246  0.87486744]
Mean of accuracy scores is for KNN Classifier is 86.67232237539766

[0.73191941 0.7397667  0.73891835 0.73276776 0.7340403  0.73488865
 0.73361612 0.73934252 0.72895016 0.73552492]
Mean of accuracy scores is for SVC Classifier is 73.49734888653234

[0.84793213 0.85387063 0.85811241 0.92746554 0.93361612 0.93679745
 0.9416755  0.94082715 0.93679745 0.93446448]
Mean of accuracy scores is for Random Forest Classifier is 91.11558854718982

[0.73828208 0.74400848 0.74209968 0.73743372 0.73616119 0.73616119
 0.73679745 0.73997879 0.73722163 0.73764581]
Mean of accuracy scores is for MultinomialNB is 73.8579003181336

[0.82757158 0.82990456 0.83372216 0.86256628 0.8678685  0.87147402
 0.87720042 0.87826087 0.87317073 0.87041357]
Mean of accuracy scores is for ADA Boost Classifier is 85.92152704135736
```

```
# Checking for difference between accuracy and mean accuracies.
lis3 = ['Logistic Regression','Decision Tree Classifier','KNeighbors Classifier','SVC', 'Random Forest Classifier',
        'MultinomialNB', 'ADA Boost Classifier']
```